Biber — the next generation backend processor for Bib\LaTeX\Xe

Philip L. Kime

Abstract
For many, particularly those writing in the humanities, Philipp Lehman’s Bib\LaTeX\Xe package has been a much welcomed innovation in\LaTeX bibliography preparation. The\LaTeX\Xe stack language and to be able to write sophisticated bibliography styles using a very rich set of\LaTeX\Xe macros is a considerable advantage. Up until 2009 however, Bib\LaTeX\Xe still relied on Bib\TeX to sort the bibliography, construct labels and to create the .bbl. The requirement for a dedicated backend processor to do such tasks was not going to go away as doing complex, fast sorting in\TeX is not a particularly amusing task. It was clear that in the future, the Bib\LaTeX\Xe backend processor needed to be able to handle full Unicode and many feature requests were being raised for things which the backend had to do and which were either impossible or nightmarish to do with Bib\TeX. Biber was created to address these issues and this article is about how it works and the many rather nice things it can do. Biber is the recommended backend processor for Bib\LaTeX\Xe, replacing Bib\TeX. There will come a time (probably around Bib\LaTeX\Xe 2) when Bib\TeX is no longer supported for use with Bib\LaTeX\Xe, so read on . . .

1 History
François Charette originally started to write Biber in 2008 and after I realised that an APA style I was writing for Bib\LaTeX\Xe required some fundamental changes to the backend processor and that Bib\TeX wasn’t going to be it (for why, see below), I had a look at the early Biber. I played with it for a while, found a small bug and submitted it. Things escalated and development entered a very rapid period where François and I knocked Biber into a releasable shape quite quickly. After a year or so, the vicissitudes of life pulled François away and I was left to my own devices with Biber gaining users rapidly, particularly in Germany, probably due to Philipp Lehman’s involvement with the development as we soon realised we had to coordinate Bib\LaTeX\Xe and Biber releases. This continues and Bib\LaTeX\Xe and Biber are now so closely linked, it is fair to say that they are essentially one product. As we approach the Bib\LaTeX\Xe 2.0 release, the plan is to drop Bib\TeX support altogether as there are so many features now which are marked “Biber only” in the Bib\LaTeX manual. It’s those features which I will describe below.

François says that the name comes from the national animal of the last country he lived in, translated into the language of the country he currently lives in. It also sounds a bit bibliographical.

2 What Biber does
Biber is used just as you would Bib\LaTeX\Xe. It’s designed to be a drop-in replacement for Bib\LaTeX\Xe users. It uses a Bib\LaTeX\Xe compatible C library called “btparse” and so existing .\bib files should work as-is. When Bib\LaTeX\Xe is told that it’s using Biber instead of Bib\TeX as the backend processor, it outputs a special .\bcf file. This is nothing more than a fancy .\aux file in XML which describes all of the necessary options, citation keys and data sources which Biber uses to construct the .\bb\Xe. XML was a natural choice as the options can get quite complex (particularly for sorting). Biber reads the .\bcf file, looks for the required data sources, reads them and looks for the citation keys also mentioned in the .\bcf. Then it constructs a .\bb\Xe and writes it. Sounds simple? It’s not. Biber is about 20,000 lines of mostly object-oriented Perl and some of the things it does are quite tricky.

3 Distributing Biber
Biber is written in Perl. This is an ideal language for such a task, as Perl 5.14 (which is what Biber uses now) has full Unicode 6.0 support and some really superb modules for collating UTF-8 which have CLDR\footnote{Common Locale Data Repository} support, allowing sorting to be tailored automatically to the idiosyncrasies of particular languages. The \text{\texttt{Text::Bib\LaTeX}} module makes parsing Bib\LaTeX files easy but I had to change the underlying \text{\texttt{btparse}} C library a little bit to make it deal with UTF-8 when forming initials out of names and to address a few other things which are the inevitable consequences of a library written probably fifteen years ago; other than that, the library has proven to be a solid foundational element of Biber. I have to thank Alberto Manuel Brando Simões, the current \text{\texttt{Text::Bib\LaTeX}} maintainer for being so flexible and releasing new versions so quickly after my hacks.

Distributing Perl programs with such module dependencies is not easy and was a major stumbling block to early adoption of Biber. Then I came across the marvellous \text{\texttt{PAR::Pack\Xe}} module which allows one to package an entire Perl tree with all dependencies into one executable which is indistinguishable from a “real” executable. One virtualised build farm later and Biber had an automated build procedure for most major platforms and was swiftly put into \TeX Live. Now all users have to do is to update their
TL installation and type “biber”. SourceForge\textsuperscript{2} is home to regular updates of the development binaries and github\textsuperscript{3} is home to the Perl source which can be used instead of the binary versions if you don’t mind installing some Perl modules (in fact, I only ever use the Perl source version myself).

4 Unicode and sorting

One of the main issues with the original BibTEX is that it is ASCII only. There is an 8-bit version bibtex8 but that’s not really enough these days. There is also a newish Unicode version bibtexu but that doesn’t help BibTEX’s myriad of other needs for its backend and it doesn’t help with CLDR and the hard problem of complex sorting.

Biber is Unicode 6.0 compliant throughout, even the file names it reads and the citation keys themselves. This means that your data sources can be pure UTF-8 which is particularly nice if you are using a UTF-8 engine like Xe\TeX\ or Lua\TeX. In fact, Biber will look at the locale settings passed by BibTEX (or those found in the environment or passed on the command line) and automatically (re)encode things to output a .bbi file in whatever encoding you want. It will even automatically convert UTF-8 to and from \TeX\ character macros/symbols in case you are using a not-quite-Unicode engine like pdf\TeX.

Sorting is one of the most important things that Biber does. Sorting the bibliography is done by default using the UCA (Unicode Collation Algorithm) via the excellent Unicode::Collate module. This is CLDR aware and so it will take notice of the locale from various sources and tailor the sort accordingly. Swedes hate it when å sorts before å and CLDR support avoids upsetting Swedes. Sorting a bibliography means dealing with sorting requirements such as:

“Sort first by name (or editor if there is no name or translator if there is no editor) and then descending by year and month (or by original year and month of publication if there is no year) and then by just the last two digits of the volume and then by title (but case insensitive for title). Oh, and if there is a special shorthand for the entry, sort by that instead and ignore everything else.”

Biber does this in complete generality using a multi-field sorting algorithm allowing case sensitivity, direction and substrings to be specified on a per-field basis. Bib\LaTeX\ defines many common sorting schemes (such as name/year/title, etc.) but you are free to define your own using a nice \LaTeX\ macro interface.

This interface makes Bib\LaTeX\ write a section in the XML .bcf which Biber reads to construct the sorting scheme it uses to sort the entries before writing the .bbi. I am not aware of any bibliography system that has better sorting but that may be wishful thinking born of spending so much time getting it to work . . .

5 Data sources and output

It may have struck readers as strange that I refer to their .bib files as “data sources”. This is because Biber can read more than just Bib\LaTeX\ format files. It has a modular data source reading/writing architecture and so new drivers can be written relatively easily to implement the ability to read new data sources and write new output formats. Data sources are read and internal entry objects constructed so that the data is processed in a source-neutral format internally. Currently, Biber can also read files in RIS format, Zotero XML/RDF format and Endnote XML format but support for these formats is experimental, partly due to weaknesses in the formats themselves, it has to be said. There is support for remote data sources for all formats by specifying a URL that returns a file in the format. This is quite useful with services such as CiteuLike which has a .bib gateway.

Biber normally outputs a .bbi file but it can also output a GraphViz .dot file which allows you to visualise your data. This is mainly useful for checking complex cross-reference inheritance and other entry-linking semantics. Biber can also output Bib\LaTeX\XML which is an experimental XML data format specially tuned for Bib\LaTeX\ (of course it can read this too).

A very nice feature of Biber is the “sourcemap” option. It is often the case that users would like to massage their data sources but they have no control over the actual source. Biber allows you to specify data mapping rules which are applied to the data as it is read, effectively altering the data stream which it sees, without changing the source itself. For example, you can:

- Drop all ABSTRACT fields as the entries are read so that their strange formatting doesn’t break \LaTeX.
- Add or modify a KEYWORD field in all BOOK or INBOOK entries which come from a data source called “references.bib” whose TITLE field matches “Collected Works” so that you can split your bibliography using Bib\LaTeX\ filters.
- Use full Perl regular expressions to match/replace in any field in the entry to regularise messy variants of a name so that the same-author disambiguation features of Bib\LaTeX\ work nicely.

Philip L. Kime
The “sourcemap” option is quite general and provides a linear mapping interface where you can specify a chain of rules to apply to each entry as it is read from the data source. The Biber PDF manual has many examples.

6 Uniqueness

A major feature is the automated disambiguation system. Depending on the options which you set in \texttt{BibLaTeX}, Biber will automatically disambiguate names by using either initials or, if necessary, full names. Even better, it can, if you like, disambiguate lists of names which have been truncated using “et al.” by expanding them past the “et al.” to the point of minimal unambiguity. (This is a requirement for APA style and the very feature I needed when I started looking at Biber. It took two years to get this implemented.) This is fairly deep magic as it interacts with name disambiguation in an unbounded loop sort of way.

The disambiguation system can be asked to do more subtle types of work too, such as disambiguating citations just enough to make them unambiguous pointers into the bibliography but not enough to make every single individual author unambiguous, etc. These are quite fine points and make sense when you read the section of the \texttt{BibLaTeX} manual which covers this, with examples. Again, I don’t know of any other bibliography system that has automated this.

7 Other features

The following features are all due to feature requests by \texttt{BibLaTeX} users and some were quite complex to implement. Some of them are waiting until \texttt{BibLaTeX 2.x} for a macro interface to expose them to users as this is when it is planned to retire \texttt{BibTeX} support from \texttt{BibLaTeX}.

- Many \texttt{BibLaTeX} options can be set on a per-entrytype basis so you can, for example, choose to truncate names lists of five or more authors with “et al.” for \texttt{BOOK} entries and choose a different limit for \texttt{ARTICLE} entries.
- Biber only needs one run to do everything, including processing multiple sections.
- You can create an entry “set” (a group of entries which are referenced/cited together) dynamically, just using \texttt{BibLaTeX} macros. With \texttt{BibLaTeX}, this requires changes to the data source.
- “Syntactic” inheritance via a new \texttt{XDATA} entry-type and field. This can be thought of as a field-based generalisation of the \texttt{BibTeX} \texttt{@STRING} functionality (which is also supported). \texttt{XDATA} entries can cascade so you can inherit specific fields defining a particular publisher or journal, for example.
- “Semantic” inheritance via a generalisation of the \texttt{BibTeX} cross-reference mechanism using the \texttt{CROSSREF} field. This is highly customisable by the user — it is possible to choose which fields to inherit for which entrytypes and to inherit fields under different names etc. Nested cross-references are also supported.
- Support for related entries, to enable generic treatment of things like “translated as”, “reprinted as”, “reprint of” etc. (\texttt{BibLaTeX 2.x})
- Customisable bibliography labels for styles which use labels (\texttt{BibLaTeX 2.x})
- Multiple bibliography lists in the same section with different sorting and filtering. (\texttt{BibLaTeX 2.x})
- No more restriction to a static data model of specific fields and entrytypes. (\texttt{BibLaTeX 2.x})
- Structural validation of the data against the data model with a customisable validation model (\texttt{BibLaTeX 2.x})

Feature requests and bug reports are always welcome via the SourceForge tracker.

⋄ Philip L. Kime
Zürich, Switzerland
Philip (at) kime dot org dot uk
http://biblatex-biber.sourceforge.net